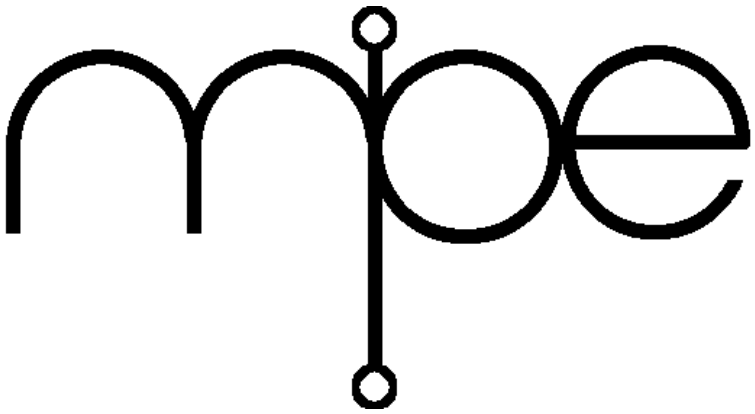
ForthEd2

A user modifiable editor

Microprocessor Engineering Limited



ForthEd2
User manual
Manual revision 2.1
5 November 2012

Software
Software version 2.1

For technical support
Please contact your supplier

For further information
MicroProcessor Engineering Limited
133 Hill Lane
Southampton SO15 5AF
UK

Tel: +44 (0)23 8063 1441
Fax: +44 (0)23 8033 9691
e-mail: mpe@mpeforth.com
tech-support@mpeforth.com
web: www.mpeforth.com

Table of Contents

1 Introducing ForthEd2

The ForthEd2 text editor in the *Examples\ForthEd2* folder is a Windows application that demonstrates using many of the facilities of VFX Forth for Windows including:

- Multi Document Interface (MDI) techniques
- Keyboard acclerators
- Printing text
- Font selection
- Resource scripts
- Edit controls
- Dialogs
- Tool tips and string tables
- Configuration files
- Multitasking
- Pipes
- DocGen project techniques
- Generating a turnkey application.

ForthEd2 can be built in two versions.

- Development version
- Application version

The development version is useful for testing and for use within a larger application. The application build generates a a turnkey application.

If you want to include ForthEd2 within your application, or to distibute it as a standalone application, you are welcome to do so, provided that you do not distibute the source code, and provided that the ForthEd2 About boxes are retained and accessible by your users from the ForthEd2 main menu. You may not distribute the ForthEd2 software manuals.

If you want to modify these terms, please contact MPE.

1.1 Development build

The development build is run by compiling *F2dev.bld*.

```
+xrefs                  \ useful when removing code
0 constant turnkey?     \ not a trurnkey app
include mainwin         \ the main file
```

To include ForthEd2 within your application, just compile *MainWin.fth* instead of *F2dev.bld*. Launch ForthEd2 by executing `RunF2`.

1.2 Application build

1.2.1 Compiling ForthEd2 as a turnkey app

The first part of the build is identical to the development build except for the setting of `Turnkey?`.

```
1 constant turnkey?     \ -- n
\ Mark the build as a turnkey build.

[undefined] fullPathName [if]
Extern: DWORD WINAPI GetFullPathName(
  char * lpFileName,
  DWORD nBufferLength,
  char * lpBuffer,
  char ** lpFilePart
);

: fullPathName  \ caddr1 len1 -- caddr2 len
\ Convert a file name to (sort of) canonical form. For example,
\ relative path names are converted to absolute path names.
\ Macro names are expanded before conversion.
  {: | ipath[ max_path 1+ ] opath[ max_path 1+ ] -- :}
  expand ipath[ zplace
  ipath[ MAX_PATH opath[ 0 GetFullPathName
  opath[ swap >SysPad
;
[then]

c" F2dir" MacroSet? 0= [if]             \ if location macro not set
  ms" %IDir%" fullPathName s" F2dir" replaces   \ use the directory containing this file
[then]

WM_USER 22 + constant USER_Unchanged    \ wparam & lparam unused

include %F2dir%\mainwin                 \ Include the main file
```

1.2.2 Creating WinMain

```
: PassOnCommandLine     \ --
```

Pass the command line on to the running instance of ForthEd2. This is only performed if another copy is already running.

Every application needs a `WinMain`. This word should be made the action of `EntryPoint`.

```
: WinMain       \ hInst hPInst lpsz nShow -- res
\ The VFX Forth cold chain has been run before WinMain is run.
  { hInst hPInst lpsz nShow | initcomctls[ initcommon ] -- res }

\ Check for a previous instance of ForthEd2 already running.
\ If it is, send the command line to it, and then exit.
  PipeExists? if
    PassOnCommandLine  0  exit
  endif

\ Initialise common controls and anything else we need.
  initcommon initcomctls[       \ start up common controls
```

```
  initcommon.dwsize !
  ICC_WIN95_CLASSES ( $0FF )
  ICC_DATE_CLASSES ( $100 ) or  ICC_USEREX_CLASSES ( $200 ) or
  ICC_COOL_CLASSES ( $400 ) or  ICC_INTERNET_CLASSES ( $800 ) or
  ICC_PAGESCROLLER_CLASS ( $1000 ) or
  initcomctls[ initcommon.dwicc !
  initcomctls[ initcommoncontrolsex drop

  RunF2                         \ start editor
\ APPIDLE exits when it receives a WM_QUIT message.
  AppIdle                       \ "Petzold" message loop

\ The exit chain will run when WinMain exits.
  ExitCode @
;
assign WinMain to-do EntryPoint
```

1.2.3 Application save

Saving the application is as usual, excpt that we turn off the requirements for the support DLL and INI files.

```
0 to InitSupport?       \ we don't need the support DLL
0 to GenINI?            \ we don't need an INI file
\ 2 Mb set-size         \ reduce memory footprint
save ForthEd2           \ save application
bye
```

1.3 Rebuilding the manual

The software manual for ForthEd2 is built by running *b.bat* in the folder *Examples\ForthEd2\Manual\DocFiles*. You may have to change the paths to VFX Forth and MikTex before use.

2 ForthEd2 text editor

The majority of the code is compiled from *MainWin.fth* which `includes` the other files. For development use, when cross references and other tools may be needed, just compile *F2dev.bld*, or to build a turnkey application just compile *F2app.bld*.

2.1 Edit controls

`: SetTopWin  \ hWnd --`
Make this the topmost window.

`: EditLen  \ hEdit -- len`
Find the number of characters in the edit box.

`: GetLine#  \ hEdit -- line#`
Get current line number.

`: Get#Lines  \ hEdit -- #lines`
Get number of lines in an edit window.

`: GetCol#  \ hEdit -- col#`
Get the current column number.

`: ReplaceSel  \ z$ hEdit --`
Replace the current selection (insert at current position if none).

`: GetCurrSel  \ buff blen hEdit -- caddr len`
Get the current selection into the given buffer, returning the selection, which may not start at the start of the given buffer.

`: GotoLine#  \ line# hEdit --`
Go to the selected line in the range 0..n-1.

`: SetTabWidth  \ n hEdit --`
Set the tab stops to every n characters.

`: BringForward  \ hwnd --`
Make this window topmost and give it the focus.

`: Redraw  \ hwnd --`
Force a window to be redrawn. Essential when adding, moving or deleting tabs in a tab control.

`NULL value hF2app  \ -- handle`
Module handle for ForthEd2 app or DLL.

`NULL value hF2  \ -- handle`
Main ForthEd2 frame window handle.

`struct /FRdata  \ -- len`
Structure for Find/Replace operations.

`0 value GFR  \ -- addr`
Returns the address of the Find/Replace structure.

`RGB_BLACK value EditForeRGB  \ -- rgb`
Editor's foreground colour.

`RGB_WHITE value EditBackRGB  \ -- rgb`
Editor's background colour.

2.2 Main Window

The identifiers and resource script are not documented. See the source code in *MainWin.fth* for the details.

`Accelerators: F2keys  \ -- addr`
The application's accelerator table.

`stringtable: TipTable  \ -- struct`
The string table for tooltip text.

`: MoveCoolbar  \ lparam --`
Move the coolbar.

2.3 Status bar

`NULL value hF2status  \ -- handle`
Status bar handle.

`0 constant LineCol#  \ -- n`
Part number for line and column display.

`1 constant spare#  \ -- n`
Part number for future use.

`2 constant FQPN#  \ -- n`
Part number for fully qualified path name

`[parts SBparts  \ -- addr`
Generates parts offsets table.

`: .NoDocsOpen  \ --`
Indicate no active docs.

`: InitStatusBar \ --`
Initialise status bar.

`: MakeStatusBar \ --`
Create and initialise the status bar.

`: MoveStatusbar \ lparam --`
Move the status bar.

2.4 Tab control

`0 value hF2tab  \ -- handle`
Tab control handle.

`#256 buffer: TabText  \ -- adr`
Buffer for text going to/from the tab control.

`create TC_SetText  \ -- addr`
Shared TCITEM structure for TCM_GETITEM and TCM_SETITEM messages.

`#26 value TabHeight  \ -- n`
Required height of the tab control.

`create TabFont  \ -- addr`
LOGFONT structure for the text in a tab control.

`create WC_TABCONTROL  \ -- addr`

Some Windows constants are actually pointers to strings. If used at run-time without the VFX support DLL, e.g. in a turnkey application, these strings will not be present and must be provided by the application. This is one of those strings.

```
: MakeTabCtrl   \ --
```
Make the tab control and initialise it with our font.

```
: MoveTabCtrl   \ lparam --
```
Move the tab control.

```
: AddTab        \ caddr len -- index
```
Add tab with given text, returning the index.

```
: DelTab        \ index --
```
Delete tab of given index.

```
: SelectTab     \ index --
```
Select the required tab.

```
: CurrTab       \ -- index|-1
```
Identify the current tab.

```
: GetTabText    \ index -- caddr len
```
Get the text from the specified tab.

```
: SetTabText    \ caddr len index --
```
Set the text for the specified tab.

```
: #Tabs         \ -- n
```
Return the number of tabs in the control.

```
: TabMatches?   \ caddr len tbuff tlen -- flag
```
Return true if the tab text matches the given string

```
: FindTab       \ caddr len -- index true | 0
```
Find the index of the tab whose text contains the given text. Trailing spaces and asterisks are ignored.

2.5 Client area

```
0 value hF2client       \ -- handle
```
MDI client window handle.

```
: GetSubmenuHandle      \ MenuRes id -- hSubMenu
```
Given a menu resource identifier *MenuRes* and a submenu identifier *id*, return the Windows handle for the submenu if it exists.

```
: MakeClient    \ --
```
Create the MDI client window.

```
: F2Height      \ -- h
```
Get height of main window client area.

```
: ClientYH      \ -- y h
```
Calculate top and height of client window.

```
: MoveClient    \ lparam --
```
Move the client window

```
: hActive       \ -- handle|0
```

Return handle of the active container window.

```
: SetActive     \ hCon --
```
Make this the active MDI child container.

2.6 Include additional files

The following files are now compiled.

```
Lib\ConfigTools.fth Generic app configuration tools
About.fth           About boxes
FileLoadSave.fth    Load and Save document files
Editing.fth         Main edit functionality
FindBox.fth         Find string dialog
GotoBox.fth         Goto line dialog
PipeCommands.fth    Commands for pipe handler
PipeServer.fth      Pipe server
EditConfig.fth      Edit configuration dialog
```

2.7 Main Winproc

```
MAX_PATH buffer: szConfig$      \ -- addr
```
Buffer holding the configuration file name as a zero-terminated string.

```
: TypeCfgText   \ --
```
Type the configuration file text.

```
: GenConfigFile \ --
```
Save the configuration file.

```
: RunConfigFile \ --
```
Load the configuration file.

```
: InitAccel     \ --
```
Initialise accelerator use.

```
: TermAccel     \ --
```
Terminate accelerator use.

```
: Calculator    \ --
```
Launch the Windows calculator.

```
: RunPrintJob   \ --
```
Print the current edit window.

```
: SendToActive  \ hwnd message wparam lparam -- res
```
Pass unprocessed messages to the active client and the default frame procedure.

```
: F2commands    \ h m w l -- status
```
Handles WM_COMMAND messages.

```
: F2notifies    { hwnd mesg wparam lparam -- ior }
```
Handles WM_NOTIFY messages.

```
: F2WinProc     { hwnd mesg wparam lparam -- ior }
```
The main winproc for the application.

```
: RunF2         \ --
```

Run ForthEd2.

```
: CloseF2       \ --
```
Close ForthEd2.

```
: FocusF2       \ --
```
Set the focus to ForthEd2.

```
: RunForthEd2   \ -- ; invoked from AIDE Utils menu.
```
If open, close it. If closed, open it.

3 About boxes

The identifiers and resource script are not documented, see *About.fth* for the source code.

```
: (AboutDialogProc)     { hdlg message wparam lparam -- ior }
```
The about box winproc is used for both boxes.

```
4 1 callback: AboutDialogProc   \ -- addr
```
The entry point for Windows callbacks.

```
: AboutF2       \ --
```
Run the "About ForthEd2" dialog.

```
: AboutMPE      \ --
```
Run the "About MPE" dialog.

4 File load and save

4.1 File Locks

```
Semaphore FileLock      \ -- addr
```
Used for locking access to the filer which uses global data.

```
: initLock      \ --
```
Initialise file locking.

```
: termLock      \ --
```
Shut down file locking.

4.2 Quick and dirty file tools

This code was developed for quick and dirty file handling during development. Note that these tools are designed for use from the keyboard and that global variables are used. The code is written for safety, not for speed. All errors cause a `THROW`. The code is derived from Wil Baden's ToolBox.

The facetious word names are entirely deliberate to remind you to remove these words from production code or to add file locking, as done here.

```
0 value pData   \ -- addr
```
Pointer to data block loaded from a file.

```
0 value /Data   \ -- n
```
Size of data block

```
0 value hData   \ -- handle
```
Handle of data file

```
: FILE-CHECK      ( n -- )  ABORT" File Access Error " ;
```
Tests a file ior.

```
: MEMORY-CHECK    ( n -- )  ABORT" Memory Allocation Error " ;
```
Tests a memory ior.

```
: rewind-file   \ file-id -- ior
```
Resets a file to the start.

```
: InitReadFile  \ handle -- size
```
Reset the file to the start and return its size.

```
: OpenMouth     \ caddr len --
```
Open the file for read only.

```
: guzzle        \ file-id -- addr length
```
Reads file from disc to HERE without ALLOTing space. The file is left open.

```
: slurp         \ file-id -- addr length
```
Reads the contents of a file into `ALLOCATE`d memory and returns the address and length. Release the memory using `BURP`. The `ALLOCATE`d memory is one byte longer than the file size.

```
: Hiccup        \ --
```
Close the file opened by `OpenMouth`.

```
: BURP          \ --
```

Release memory ALLOCATEd by SLURP.

```
: Inhale        \ fname flen -- caddr len
```
Given a file name, reads the file and a trailing zero into ALLOCATEd memory, returning its address and length. Use in the form:

```
s" <filename>" INHALE ... BURP
```

```
: Suck          \ "<filename>" -- caddr len
```
Reads the given file into ALLOCATEd memory, returning its address and length. Use in the form:

```
SUCK <filename> ... BURP
```

```
: Exhale        \ caddr len name namelen --
```
Write the memory region defined by caddr/len to the file whose name is given by name/namelen. A THROW occurs on any error.

```
: Vomits        \ caddr len "<filename>" --
```
Like Exhale, but takes the filename from the input stream.

```
VOMITS <filename>
```

4.3 File loading

```
MAX_PATH cell + buffer: szFileDir       \ -- addr
```
Default file load/save directory.

```
MAX_PATH cell + buffer: szLastFile      \ -- addr
```
Last file loaded/saved.

```
MAX_PATH cell + buffer: szInsertDir     \ -- addr
```
Default file insert directory.

```
MAX_PATH cell + buffer: szInsertFile    \ -- addr
```
Last file inserted.

```
: LoadFile      \ caddr len hEdit -- ior
```
Load a file into an edit control at the current selection point. If a selection has been made, it is replaced.

```
: SaveFile      \ caddr len hEdit -- ior
```
Save contents of the edit box to the given file.

```
: SelLoadFile   \ -- caddr len true | false
```
Use the file selector dialog to get a file name. If no file is selected, only false (0) is returned.

```
: SelInsertFile \ -- caddr len true | false
```
Use the file selector dialog to get a file name. If no file is selected, only false (0) is returned.

```
: SelSaveFile   \ -- caddr len true | false
```
Use the file selector dialog to get a file name. If no file is selected, only false (0) is returned.

4.4 Initialisation and termination

```
: InitFiles     \ --
```
Initialise use of the filer.

```
: TermFiles     \ --
```
Shut down use of the filer.

```
: SaveCfgFiler  \ --
```
Save the filer configuration in the configuration file.

5 Editing functions

Edit windows consist of a container handling a RichEdit control. We do not use the VFX Richedit device as it has already been subclassed for console use. With the use of keyboard accelerators we can avoid subclassing the control for now.

When an edit window is created, an edit information structure is aded to a linked list.

```
: EditError     \ z$ --
```
Display editing error message.

```
0 value ShowEditErrors? \ -- flag
```
True to enable some messages

```
: ?EditError    \ z$ --
```
Display editing error message if enabled.

5.1 Editor data

```
0 value hEditFont       \ -- handle
```
Handle of font used by edit controls.

```
create EditFont \ -- addr
```
LOGFONT structure for the text in an edit control

```
variable EditList       \ -- struct
```
Anchors list of edit information structures.

```
8 value /DefTabs        \ -- n
```
Default tab size in characters.

```
1 value RestoreFiles?   \ -- n
```
Set non-zero to restore files at startup.

```
1 value SaveFiles?      \ -- n
```
Non-zero to save files during an edit window close.

```
1 value ViewMode        \ -- id
```
The current view mode. Mode numbers are:

```
  1 constant #Cascade   \ -- n
  2 constant #TileH     \ -- n
  3 constant #TileV     \ -- n
  4 constant #Maximise  \ -- n
```

```
cell +USER MyInfo       \ -- addr
```
User variable holding info address in winprocs.

```
struct /EditInfo        \ -- size
```
Defines an edit information structure.

```
  int ei.*Next                  \ pointer to next window: MUST BE FIRST
  int ei.hContainer             \ hwnd of container
  int ei.hEdit                  \ hwnd of editbox inside the container
  int ei.flags                  \ Flags: see below
  int ei.short                  \ offset into file name for short name
  MAX_PATH field ei.FileName    \ zero terminated file/window name
```

```
end-struct

$0001 constant HAS_FILENAME     \ filename has been set
$0002 constant HAS_CHANGED      \ text has changed
$0004 constant HAS_EDIT_WINDOW  \ edit control created
$0008 constant IS_READONLY      \ file is read only
```

`: ei.ShortName  \ info -- zshort`
Return the short name.

5.2 General `Info` functions

`: SetFlag       \ mask info --`
Get the mask bits in the `ei.Flags` field.

`: ClrFlag       \ mask info --`
Clear the mask bits in the `ei.Flags` field.

`: Flagged       \ mask info -- flag`
Test the mask bits in the `ei.Flags` field.

`: NewInfo       \ -- struct|0`
Create and add a new info structure to the list.

`: DelInfo       \ struct --`
Remove edit info structure from the list and delete it.

`: #info         \ -- n`
Find the number of info structures.

`create szInfoProp       \ -- zaddr`
The string used for the info structure property of container and edit windows.

`: SetInfoProp   \ info handle --`
Give the window a pointer to its info structure.

`: GetInfoProp   \ handle -- info|0`
Get the address of the window's info structure.

`: hEdit         \ -- hEdit|0`
Return the handle of the active edit control.

`: hCon          \ -- hCon|0`
Return the handle of the active container control.

`: FocusEdit     \ info --`
Select edit control for focus.

`: FileEdit      \ info --`
Put full file name in status bar.

`: InfoTab       \ info -- index true | 0`
Find the tab for the edit control.

`: CloseTab      \ info --`
Close the tab for this edit control.

5.3 Status bar display

```
: LineInfo      \ hEdit --
```
Display line and column numbers on the statusbar

5.4 Edit window creation

```
create zNew     \ -- zaddr
```
Base name for a new window.

```
  constant EDIT_STYLE   \ -- style
```
Default style for the edit controls.

```
: MakeEdit      \ info --
```
Make an edit box, using and filling in the info structure as it goes.

```
: CloseEdit     \ info --
```
Close the associated edit control.

```
: GetEventMask  \ hEdit -- mask
```
Get the Windows event mask for the window.

```
: SetEventMask  \ mask hEdit --
```
Set the Windows event mask for the window.

```
: +InfoMask     \ mask info --
```
Set the event mask in the edit control for the structure.

```
: -InfoMask     \ mask info --
```
Clear the event mask in the edit control for the structure.

```
: +Changes      \ info --
```
Enable EN_CHANGE messages from the edit control.

```
: -Changes      \ info --
```
Disable EN_CHANGE messages from the edit control.

```
: doUnchanged   \ xt info --
```
perform *xt* with EN_CHANGE messages disabled for the *info* block.

```
: doWinUnchanged        \ xt hwnd --
```
Perform *xt* with EN_CHANGE messages disabled for the window.

```
: +DefNotifies  \ info --
```
Set the default notifications from the edit control.

5.5 Loading and saving files

```
: SetTitleBar   \ z$ info --
```
Set the container's title bar.

```
: +Changed      \ caddr len -- caddr len'
```
Add an asterisk '*' change marker to the string.

```
: InfoChanged   \ info --
```
Change the edit window's container title and tab to show that it has changed.

```
: InfoUnchanged \ info --
```
Change the edit window's container title and tab to show that it is unchanged.

```
: ShortLen      \ caddr len -- len'
```

Find the length after a directory separator.

```
: SetFilename   \ caddr len info --
```
Set the name and short offset for the given file name.

```
: ?GoodSave     \ flag info -- ; flag=0 for success
```
If flag is zero for success, mark the structure as having a file name.

```
: SaveEditAs    \ info --
```
Open a file selector dialog and save the file if the user selected a file.

```
: (SaveEdit)    \ info --
```
Save the contents of the edit box. The file name must be valid.

```
: SaveEdit      \ info --
```
Save the contents of the edit box. If a filename has already been set it is used, otherwise the user is prompted for one.

```
: Save?         \ info -- flag ; true for yes
```
Ask user if the file should be saved.

```
: ?SaveEdit     \ info --
```
Only save file if it has changed and `SaveFiles?` is. non-zero. Prompt the user to save the file. If it already has a name, use it, otherwise put up a dialog.

5.6 Container handling

```
0 value StillMaking?    \ -- flag
```
Set true while the windows are being created to avoid message sequencing problems.

```
: ResizeContainer       \ h m w l -- status
```
Resize the container and edit control.

```
: ContainerCommands     \ h m w l -- status
```
Process WM_COMMAND messages for the container and edit control.

```
: ContainerFocus        \ hCon -- status
```
Focus is always passed to the edit control, and the status bar is updated.

```
: RedrawEdits   \ --
```
Redraw all edit windows.

```
: ContainerNotify       \ h m w l -- status
```
Process WM_NOTIFY messages for the container and edit control.

```
: CloseContainer        \ h m w l -- status
```
Close the container and edit control.

```
0 value WM_FindMessage  \ -- msg
```
Message number for find/replace

```
defer HandleFindReplace         \ *fr --
```
Place holder for Find/Replace word

```
: (ContainerProc)       \ h m w l -- sctatus
```
The winproc for containers and their edit windows.

```
4 1 callback: ContainerProc     \ -- addr
```
The entry point for Windows callbacks to containers' winprocs.

```
create szContainer      \ -- addr
```

Class name for container windows.

`create ContainerTemplate \ -- addr`
An MDICREATESTRUCT structure used when creating new container windows.

`constant ClassStyle \ -- style`
The style used by containers.

`create ContainerClass \ -- addr`
The WNDCLASS structure used for container windows.

`: MaxSize? \ -- style|0`
If the current window is maximised, return WS_MAXIMIZE, otherwise return 0.

`: MakeContainer \ info --`
Build the the container for the edit control.

5.7 Actions for MDI frame window

`: .NoActive \ --`
Show a "no active document" message.

`0 value #new \ -- n`
Number of new document windows already opened.

`: NewName \ zbuff --`
Generate the next new document name string in the buffer as a zero terminated string.

`: SetMaximise \ --`
Set the display mode to maximised.

`: MakeDoc \ info --`
Make a new document's container and edit control, and initialise them.

`: NewDoc \ --`
Make a new document with a "New" title.

`: LoadDoc \ info --`
Load a file into a document.

`: DocOpen? \ caddr len -- info|0 ; true if open`
Check if file is already loaded.

`: SetInfoUnChanged \ info --`
Mark the associated windows as unchanged by posting a message.

`: (OpenDoc) \ caddr len --`
Create a new document and load the file into into it.

`: OpenExisting \ caddr len --`
If the specified file exists, create a new document and load the file into it.

`: OpenDoc \ --`
Run an open-file dialog and load the document.

`: LocateDoc \ line# caddr len --`
If the file has not been loaded, open and load a new document. Go to the requested line. The line number is 1 based.

`: OpenDropFiles \ hdrop --`
Open documents that have been requested by "drag and drop".

`: InsertDoc \ --`
Run an open-file dialog and load the document.

`: SaveDoc \ --`
Save the current document.

`: SaveDocAs \ --`
Save the current document with a new name.

`: CloseDoc \ --`
Close the current document.

`: CloseAllDocs \ --`
Close all open documents.

`: ?SaveAllDocs \ --`
Save all changed open documents, with prompting.

`: Tab>Info \ index -- info|0`
Given a tab index, find the document's info structure by comparing names.

`: TabChanged \ lparam --`
When the user selects a tab, its document is brought to the top.

`: SendActive \ mesg --`
If an active window exists, send the message to its edit control with WPARAM and LPARAM set to zero.

`: SetCascade \ --`
Set the display to cascade mode.

`: SetArrange \ --`
Tidy up the icons for minimised edit windows.

`: SetMaximise \ --`
Set the display mode to maximised.

`: SetMaximise \ --`
Set the display mode to maximised. This is done by maximising all the current windows.

`: SetTileH \ --`
Set the display mode to horizontally tiled.

`: SetTileV \ --`
Set the display mode to vertically tiled.

`: SetViewMode \ mode# --`
Set the view mode according to one of the view numbers.

5.8 Initialisation and Termination

`: InitEdit \ --`
Initialisation required before using edit windows and their containers.

`: TermEdit \ --`
Shutdown required after using edit windows and their containers.

`: GetCharSize { hwnd | tm[ TEXTMETRIC ] -- charW charH }`
Given the handle of a window, GetCharData returns the width and height of the character in Windows logical units. These correspond to the units returned by the GetCaretPos API call. It is assumed that the window is using the ANSI_FIXED_FONT.

6 Find text box

The find/replace system uses the standard Windows dialogs.

6.1 Global data

The Find and Replace dialog boxes are owned by ForthEd. They are passed a global FindReplace structure to permit using the F3 key to repeat the search later. The global structure is `ALLOCATE`d at editor startup.

`struct /FRdata  \ -- len`
Structure for Find/Replace operations.

`0 value GFR     \ -- addr`
Returns the address of the global Find/Replace structure. This is `ALLOCATE`d before use.

`0 value hFind   \ -- hdlg`
Zero until the find dialog has been run once.

6.2 Dialog set up

`: initFindData  \ info --`
Initialise the find/replace structure.

`: AfterRunFind  \ hdlg --`
Mark the dialog as modeless.

`: RunFindText   \ info --`
Run the Find dialog for the given document.

`: RunReplaceText        \ info --`
Run the Replace dialog for the given document.

6.3 Find message handling

This section makes heavy use of the Windows FINDREPLACE structure. See MSDN or its online documentation (on the VFX Forth Help menu) for more details. The parameters labelled opt correspond to the **Flags** member of the FINDREPLACE structure.

`: FindNextText  \ *fr opt info -- flag ; 0=not found`
Find the next text item using the FINDREPLACE structure **fr*, *opt* contains flags, and *info* is a document information structure. If the text is found, non-zero is returned.

`: RepeatFindNext        \ --`
Find the next text item. Used when the F3 key is pressed.

`: ReplaceCurrText       \ *fr info --`
Perform the replace action.

`: AtSel?        \ *fr opt info -- flag ; 0=no`
Return true if selected text matches Find text.

`: GotoFindText  \ *fr opt info -- flag`
Go to the selected text.

`: (HandleFindReplace)   \ *fr --`
When a FINDMSGSTRING message is received, this word handles it.

6.4 Initialisation and termination

```
create FINDMSGSTRING    \ -- addr
```
The string used to identify Find/Replace messages. It is used to find the actual message number.

```
: InitFind      \ --
```
Initialise ForthEd2's find/replace mechanism.

```
: TermFind      \ --
```
Shut down ForthEd2's find/replace mechanism.

7 Go to line box

The identifiers and resource script are not documented, see *\i{GotoBox.fth) for the source code.

```
0 value hGotoLineDlg    \ -- handle
```
Dialog handle.

```
: ProcessGoto   \ hdlg --
```
Goto the selected line if ok.

```
: (GotoDlgProc) { hdlg message wparam lparam -- ior }
```
The dialog box's winproc.

```
4 1 callback: GotoDlgProc       \ -- addr
```
The Windows winproc's entry point.

```
: RunGotoDlg    \ --
```
Run the Goto Dialog box.

8 Editor configuration

The identifiers and resource script are not documented, see *EditConfig.fth* for the source code.

8.1 Font handling

```
0 value hTempFont       \ -- handle
```
Temporary font handle.

```
create TempFont \ -- addr
```
Temporary LOGFONT structure for displayed text.

```
  constant DefFlags     \ -- flags
```
Default font flags

```
SCREEN_FONTTYPE constant DefType        \ -- type
```
Default font type.

```
: initTempFont  \ --
```
Copy the current edit font and handle to scratch versions.

```
: ?DelTempFont  \ --
```
If the new font is not the same as the current edit font, delete it.

```
: ChooseEditFont        \ --
```
Run the Windows Font selector dialog for the edit window font. We need to use a scratch buffer because the font dialog return data may be thrown away if the Apply button is not used.

```
: NewEditFont   \ hFont --
```
Apply the given font to all open edit windows and make it the default for new edit windows.

8.2 Dialog handling

```
: ApplyEditFont \ flag hDlg -- flag' hDlg
```
Apply the selected font. If the selection is invalid the flag is cleared.

```
: ApplyTabSize  \ flag hDlg -- flag' hDlg
```
Apply the selected tab width. If the selection is invalid the flag is cleared.

```
: ApplyRestore  \ flag hDlg -- flag' hDlg
```
Apply the restore state. If the selection is invalid the flag is cleared.

```
: ApplyEditCfgDlg       \ hdlg --
```
Apply the edit dialog configuration.

```
: SetFontName   \ hdlg --
```
Set the font name box.

```
: InitEditCfgDlg        \ hdlg --
```
Initialise the controls in the dialog.

```
: (EditCfgDlgProc)      \ hdlg message wparam lparam -- ior
```
The dialog's winproc action.

```
4 1 callback: EditCfgDlgProc    \ -- addr
```
The dialog's Windows entry point.

```
: ConfigureEditing      \ --
```
Runs the edit configuration dialog.

8.3 Save and load configuration

These actions are required for the configuration file.

```
: InstallEditFont       \ --
```
Create the previously specified font.

```
: ?ReloadDoc    \ line# caddr len --
```
If files are to be restored, reload the document.

```
: GenCfgInfo    \ info --
```
Generate the configuration line

```
  <line#> s\" <filename>" ?ReloadDoc
```

```
: SaveEditCfg   \ --
```
Save the editor configuration.

9 Pipe and command line processing

Command strings sent to the pipe server are interpreted as Forth source. The pipe server is also used for command line processing.

9.1 Command line processor

```
: skipToken     \ caddr len -- caddr' len'
```
Step over the next token in the string. If the token starts with a '"' it is treated as a string.

```
: doCommand$    \ caddr len --
```
Process the string as a Windows command line

```
: ProcessCommandLine    \ --
```
Get the Windows command line and process it

9.2 Pipe commands

```
variable LocLine#       \ addr --
```
Line number for locate commands.

```
: -l            \ -- ; -l <line#>
```
Read decimal line number and sets it as the line number for the next -F command.

```
: -f            \ -- ; -f <filename>
```
Takes any previously set line number, reads filename from the input stream and opens the file. If you use ForthEd2 as the default editor for VFX Forth, the required locate command string is:

```
  -L %L% -F "%F%"
```

```
: -cl           \ -- ; -cl <command line>
```
The following text is a Windows command line. Process it in the same way as this instance's command line.

10 Named Pipe Server and Client

ForthEd2 sets up a named pipe `\\.\pipe\ForthEd2` which can be used by other applications to send commands to the editor. To avoid timing problems, the pipe server is also used for some internal processing such as command line processing.

10.1 Pipe access primitives

```
create z$F2pipe \ -- addr
```
Name of ForthEd2's pipe as a zero-terminated string.

```
: PipeExists?   \ -- x ; nz if exists
```
Check if the pipe exists, returning nonzero if it exists.

```
: #Pipes        \ hPipe -- n
```
Find out how many instances of the pipe exist.

```
: PipeMessage?  \ hPipe -- n
```
Returns the number of characters waiting to be read from the pipe.

```
#10 constant /PipeSleep \ -- ms
```
The polling interval when waiting for data to arrive at a pipe.

```
: WaitPipe      \ hPipe --
```
Wait until the pipe receives a message. This does not return the number of bytes because more might have arrived by the time you read the pipe.

10.2 Pipe Server

Pipes are created as message pipes, not as byte streams. A consequence of this is that all output must be handled by one write operation. In order to prevent race problems in applications, all pipe transactions are handled as follows:

- Client and server open/create pipe instances,
- Client sends Forth source text to the server,
- Server `EVALUATE`s the source,
- Depending on whether there was a `THROW` during the evaluation, the server returns the string `" !Error! "` or `" OK "`,
- Client and server close their instances.

Providing that clients wait until an instance becomes available, it is not necessary for the server to be multi-threaded. All transactions are handled by one task and handling of transactions is purely sequential. This simplifies problems in applications such as database servers.

So that other applications can find the pipe, there must always be one open instance of the pipe. A new instance is created before each transaction starts, and the current instance is closed when the transaction finishes. When an application starts, it can check whether a previous instance is already running by checking for the existence of of the application's named pipe.

```
0 value hNextPipe       \ -- hPipe
```
The handle of the next pipe instance opened.

```
0 value hCurrPipe       \ -- hPipe
```
The handle of the current pipe instance being processed.

```
0 value ClosePipe?      \ -- flag
```
The pipe server task inspects this flag to see if it should terminate. Just before it terminates itself, it resets this flag to acknowledge the close request.

```
0 value IpPB    \ -- addr
```
The pipe server input buffer.

```
0 value OpPb    \ -- addr
```
The pipe server output buffer.

```
0 value PipeTib \ -- addr
```
The pipe server's terminal input buffer.

```
#4096 constant /PipeBuffs       \ -- len
```
The requested size of pipe data buffers.

```
#4096 constant /PipeTib \ -- len
```
The requested size of pipe terminal input buffer.

```
: PipeErr       \ z$ --
```
Put up an error box with a message.

```
: MakeF2pipe    \ -- hPipe|0
```
Create a named pipe for ForthEd2. Zero is returned on error.

```
task PipeTask   \ -- taskid
```
The pipe server task's control block.

```
: SetupPipes    \ --
```
Set up the resources needed by the pipe server. Used by the pipe server task.

```
: ShutPipes     \ --
```
Delete the resources needed by the pipe server. Used by the pipe server task.

```
: ProcessPipe   \ --
```
Check and process any pipe input. If you want to handle multiple lines of input text, increase `/PipeBuffs` and replace the use of `EVALUATE` with `IncludeMem`. If you want to return more than just an acknowledgement, use a buffer device such as that in *Lib\Genio\Buffer.fth* as the output device, and return it.

```
: PipeServer    \ 0 -- ior
```
The action of the pipe server task.

```
: InitPipeServer        \ --
```
Start the pipe server, waiting until it is active.

```
: TermPipeServer        \ --
```
Stop the pipe server, waiting until it is inactive.

10.3 Pipe Clients

```
#1000 value XchgMs      \ -- ms
```
Maximum time for a transaction on the same machine.

```
: ClientXchg    \ z$ op olen ip ilen -- ip ilen' 0 | nz
```
Send the output string *op/olen* to the server, and receive the response in the buffer *ip/ilen*. If the exchange succeeds, the `ip/ilen'` is the returned string and zero is returned. If the exchange fails, just a non-zero result is returned. If the input buffer is too small, the exchange fails.

10.4 Test code

```
#256 buffer: ipBuff     \ -- addr
#256 buffer: opBuff     \ -- addr

: tx            \ -- ip ilen' 0 | nz
  s" hooray" ipbuff swap cmove
  s" foo bar" dup >r opbuff swap cmove
  z$F2pipe opbuff r> ipBuff #256 ClientXchg
;

: ty            \ -- ip ilen' 0 | nz
  s" hooray" ipbuff swap cmove
  s" words" dup >r opbuff swap cmove
  z$F2pipe opbuff r> ipBuff #256 ClientXchg
;
```

Index

F

G

H

I

L

M

N

O

P

R

S

T

V

W

X

Z